

# Projeto e Análise de Algoritmos

## NP Completude

Prof. Humberto Brandão  
[humberto@bcc.unifal-mg.edu.br](mailto:humberto@bcc.unifal-mg.edu.br)

Prof. Douglas Castilho  
[douglas@bcc.unifal-mg.edu.br](mailto:douglas@bcc.unifal-mg.edu.br)

# Introdução

- Problemas **intratáveis ou difíceis são comuns** na natureza e nas áreas do conhecimento;

# Introdução

- Problemas **intratáveis ou difíceis são comuns** na natureza e nas áreas do conhecimento;
- Problemas “**fáceis**”:
  - resolvidos por **algoritmos polinomiais**;

# Introdução

- Problemas **intratáveis ou difíceis são comuns** na natureza e nas áreas do conhecimento;
- Problemas “**fáceis**”:
  - resolvidos por **algoritmos polinomiais**;
- Problemas “**difíceis**”:
  - **no momento**, conhecemos apenas **algoritmos exponenciais** para resolvê-los;

# Introdução

- **Polinomial:**
  - função de complexidade é  $O(p(n))$ , onde  $p(n)$  é um polinômio.

# Introdução

- **Polinomial:**
  - função de complexidade é  $O(p(n))$ , onde  $p(n)$  é um polinômio.
  - Por exemplo:
    - **pesquisa binária ( $O(\log n)$ );**
    - pesquisa seqüencial ( $O(n)$ );
    - ordenação por inserção ( $O(n^2)$ );
    - multiplicação de matrizes ( $O(n^3)$ );

# Introdução

- **Exponencial:**
  - função de complexidade é  $O(c^n)$ ,  $c > 1$ ;

# Introdução

- **Exponencial:**
  - função de complexidade é  $O(c^n)$ ,  $c > 1$ ;
  - Por exemplo:
    - Problema do caixeiro viajante (PCV);
    - Problema de localização de Facilidades;
    - Problema de Mochila;
    - Problema de Roteamento de Veículos.

# Introdução

- **Exponencial:**
  - função de complexidade é  $O(c^n)$ ,  $c > 1$ ;
  - Por exemplo:
    - Problema do caixeiro viajante (PCV);
    - Problema de localização de Facilidades;
    - Problema de Mochila;
    - Problema de Roteamento de Veículos.
  - Mesmo problemas de pequeno e médio porte não podem ser resolvidos por algoritmos não-polinomiais.

# Classe de Problemas: NP-Completo (Introdução)

- A teoria de complexidade a ser apresentada **não mostra como obter algoritmos polinomiais** para problemas que demandam algoritmos exponenciais, **nem afirma que não existem**;

# Classe de Problemas: NP-Completo (Introdução)

- A teoria de complexidade a ser apresentada **não mostra como obter algoritmos polinomiais** para problemas que demandam algoritmos exponenciais, nem afirma que **não existem**;
- É possível mostrar que os **problemas** para os quais **não há algoritmo polinomial** conhecido são **computacionalmente relacionados**.

# Classe de Problemas: NP-Completo (Introdução)

- Estes problemas **formam a classe conhecida como NP-Completo;**

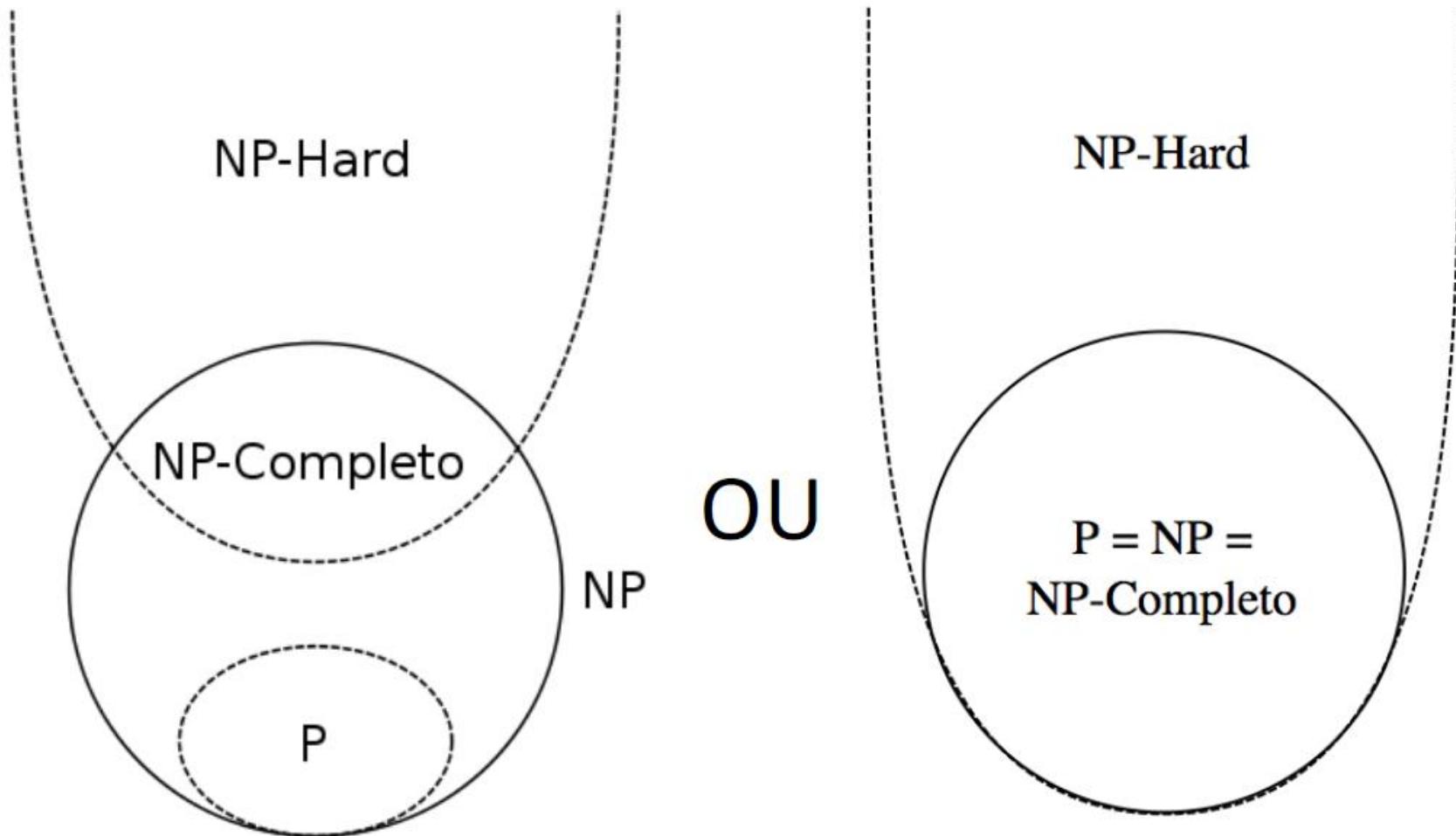
# Classe de Problemas: NP-Completo (Introdução)

- Estes problemas **formam a classe conhecida como NP-Completo;**
- Propriedade:
  - *um problema da classe NP-Completo poderá ser resolvido em tempo polinomial se e somente se todos os outros problemas em NP também puderem;*

# Classe de Problemas: NP-Completo (Introdução)

- Estes problemas **formam a classe conhecida como NP-Completo;**
- Propriedade:
  - *um problema da classe NP-Completo poderá ser resolvido em tempo polinomial se e somente se todos os outros problemas em NP também puderem;*
- Este fato é um indício forte de que dificilmente alguém será capaz de encontrar um algoritmo eficiente para um problema da classe NP-Completo.

# Classe de Problemas: NP-Completo (Introdução)



# Problemas Fáceis vs. Difíceis

# Problemas Difíceis vs. Problemas Fáceis

- 1a) Dado um grafo  $G(V,A)$  e um inteiro  $k$ , existe um caminho entre vértices  $u,v$  de tamanho no máximo  $k$ ?
  
- 1b) Dado um grafo  $G(V,A)$  e um inteiro  $k$ , existe um caminho entre vértices  $u,v$  de tamanho no mínimo  $k$ ?

# Problemas Difíceis vs. Problemas Fáceis

1a) Dado um grafo  $G(V,A)$  e um inteiro  $k$ , existe um caminho entre vértices  $u,v$  de tamanho no máximo  $k$ ?

Fácil (caminho mais curto)

1b) Dado um grafo  $G(V,A)$  e um inteiro  $k$ , existe um caminho entre vértices  $u,v$  de tamanho no mínimo  $k$ ?

Difícil (caminho mais longo)

# Problemas Difíceis vs. Problemas Fáceis

- 3) Dado um grafo  $G(V,A)$ , existe um ciclo simples que passa todos os vértices sem repetir nenhum?

# Problemas Difíceis vs. Problemas Fáceis

3) Dado um grafo  $G(V,A)$ , existe um ciclo simples que passa todos os vértices sem repetir nenhum?

3a ) grau máximo igual a 2

Fácil

3b) grau máximo > 2

Difícil (problema do ciclo hamiltoniano)

# Problemas Difíceis vs. Problemas Fáceis

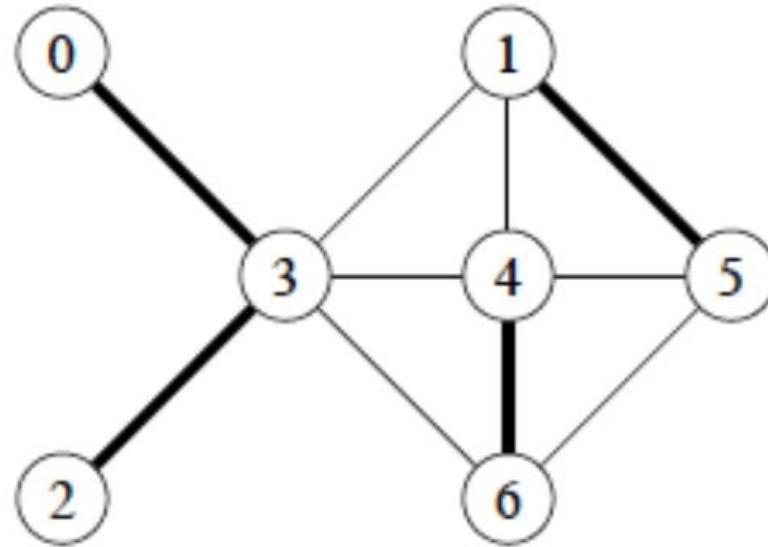
5a) Dado um grafo  $G(V,A)$  e um inteiro  $k$ , existe uma cobertura de arestas de tamanho no máximo  $k$ ?

Uma cobertura de arestas de um grafo  $G = (V,A)$  é um subconjunto  $A'$  de  $A$  tal que todo vértice de  $V$  é parte de pelo menos uma aresta de  $A'$

5b) Dado um grafo  $G(V,A)$  e um inteiro  $k$ , existe uma cobertura de vértices de tamanho no máximo  $k$ ?

Uma cobertura de vértices de um grafo  $G = (V,A)$  é um subconjunto  $V'$  de  $V$  tal que toda aresta de  $A$  é incidente em pelo menos um vértice de  $V'$

# Problemas Difíceis vs. Problemas Fáceis



Cobertura de Arestas:  $A' = \{ (0,3), (0,2), (4,6), (1,5) \}$

Cobertura de Vértices:  $V' = \{3, 4, 5\}$

# Problemas Difíceis vs. Problemas Fáceis

5a) Dado um grafo  $G(V,A)$  e um inteiro  $k$ , existe uma cobertura de arestas de tamanho no máximo  $k$ ?

Fácil

5b) Dado um grafo  $G(V,A)$  e um inteiro  $k$ , existe uma cobertura de vértices de tamanho no máximo  $k$ ?

Difícil

# Problemas NP e NP-Completo

# NP-Completo e os Problemas de Decisão

- Problemas de Decisão (Sim ou Não)
  - Arcabouço teórico
- Mas muitos problemas são de otimização...
  - problemas de otimização vs. De decisão
    - Determine o circuito simples que passa por todos os vértices com custo total mínimo (otimização)
    - Existe um circuito simples que passa por todos os vértices com custo menor ou igual a  $k$ ? (decisão)
  - Problema de decisão não é mais difícil que de otimização
    - Se temos evidência que problema de decisão é difícil, provavelmente o problema de otimização relacionado também será

# NP-Completo e os Problemas de Decisão

- **Característica fundamental da classe NP-Completo:** problemas “sim/não” para os quais uma dada solução pode ser verificada facilmente.

# NP-Completo e os Problemas de Decisão

- **Característica fundamental da classe NP-Completo:** problemas “sim/não” para os quais uma dada solução pode ser verificada facilmente.
- *A solução pode ser muito difícil de ser obtida, mas uma vez conhecida ela pode ser verificada em tempo polinomial.*

# Classe NP

Problemas de decisão cuja solução pode ser **verificada** em tempo polinomial com **algoritmo determinista**

ou

Problemas de decisão cuja solução pode ser **determinada** em tempo polinomial com **algoritmo não determinista**

# Algoritmos Não-Determinísticos

# Algoritmos Não-Determinísticos

- Antes de falar de *não-determinismo*, vamos definir os Algoritmos determinísticos:

# Algoritmos Não-Determinísticos

- Antes de falar de *não-determinismo*, vamos definir os Algoritmos determinísticos:
  - Algoritmos Determinísticos: o resultado de cada operação é definido de forma única;

# Algoritmos Não-Determinísticos

- Os algoritmos podem conter operações cujo resultado não é definido de forma única;

# Algoritmos Não-Determinísticos

- Os algoritmos podem conter operações cujo resultado não é definido de forma única;
- **Algoritmo não-determinístico:** capaz de escolher uma dentre as várias alternativas possíveis a cada passo;

# Algoritmos Não-Determinísticos

- Os algoritmos podem conter operações cujo resultado não é definido de forma única;
- Algoritmo não-determinístico: capaz de escolher uma dentre as várias alternativas possíveis a cada passo;
- Algoritmos não-determinísticos contêm operações cujo resultado não é unicamente definido, ainda que limitado a um conjunto definido de possibilidades.

# Algoritmos Não-Determinísticos

- Os algoritmos podem conter operações cujo resultado não é definido de forma única;
- Algoritmo não-determinístico: capaz de escolher uma dentre as várias alternativas possíveis a cada passo;
- Algoritmos não-determinísticos contêm operações cujo resultado não é unicamente definido, ainda que limitado a um conjunto definido de possibilidades.
- **Vamos definir uma função irreal** para os algoritmos não determinísticos...

# Algoritmos Não-Determinísticos

## Função *Escolhe*

Função **escolhe (C)** onde C é um conjunto de alternativas

- Escolhe a alternativa que leva à solução ótima, caso uma exista
- Caso não exista solução ótima, escolhe pode retornar qualquer resposta
- Complexidade: O(1)
  
- **Arcabouço teórico: não se esqueça disto!!!**

# Algoritmos Não-Determinísticos

## Função *Escolhe*

- Pesquisar um elemento  $x$  em um vetor  $A$  de  $n$  elementos

```
void PesquisaND(A, 1 , n)
{ j ← escolhe(A, 1 , n)
  if (A[ j ] == x) sucesso; else insucesso;
}
```

- Algoritmo não determinista :  $O(1)$
- Pesquisa sequencial é  $\Omega(n)$ , deterministicamente

# Algoritmos Não-Determinísticos

## Função *Escolhe*

- Problema da Satisfabilidade

```
void AvalIND(E, n);
{ for (i = 1; i <= n; i++)
    {  $x_i \leftarrow$  escolhe (true, false);
      if ( $E(x_1, x_2, \dots, x_n) ==$  true) sucesso; else insucesso;
    }
}
```

- Algoritmo não determinista :  $O(n)$
- Solução determinística:  $O(2^n)$

# Algoritmos Não-Determinísticos

## Função *Escolhe*

- Uma máquina capaz de executar a função *escolhe* admite a capacidade de **computação não-determinística**.

# Algoritmos Não-Determinísticos

## Função *Escolhe*

- Uma máquina capaz de executar a função *escolhe* admite a capacidade de computação não-determinística.
- **Uma máquina não-determinística é capaz de produzir cópias de si mesma** quando diante de duas ou mais alternativas, e continuar a computação independentemente para cada alternativa.

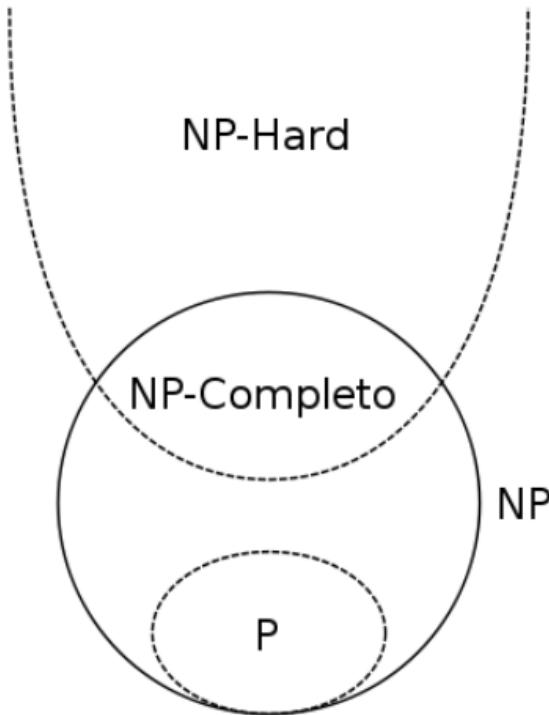
# Algoritmos Não-Determinísticos

## Função *Escolhe*

- Uma máquina capaz de executar a função *escolhe* admite a capacidade de **computação não-determinística**.
- Uma máquina não-determinística é capaz de produzir cópias de si mesma quando diante de duas ou mais alternativas, e continuar a computação independentemente para cada alternativa.
- A máquina não-determinística que acabamos de definir não existe na prática, mas ainda assim fornece fortes evidências de que certos problemas não podem ser resolvidos por algoritmos determinísticos em tempo polinomial.

# A Classe NP

# Classe NP



Classe NP inclui problemas de decisão que:

- podem ser verificados em tempo polinomial deterministicamente, ou
- podem ser resolvidos em tempo polinomial não deterministicamente (usando a escolha)

Classe P inclui problemas de decisão que:  
podem ser resolvidos em tempo polinomial deterministicamente

$P \subset NP$  ou  $P = NP$  ??

# A Classe NP-Completo

O Célebre  
Problema da Satisfabilidade Booleana (SAT)

# SAT

## O tamanho do espaço de buscas pode ser grande demais

- SAT: Encontrar um conjunto de valores para variáveis booleanas de forma a avaliar uma expressão booleana qualquer como VERDADEIRA.

## O tamanho do espaço de buscas pode ser grande demais

- SAT: Encontrar um conjunto de valores para variáveis booleanas de forma a avaliar uma expressão booleana qualquer como VERDADEIRA.
- Suponha a expressão:

$$F(x) = \overline{(x_{89} \vee x_{78} \wedge x_{01})} \wedge \overline{(x_{10} \vee x_{99})} \vee \dots \wedge (x_{56} \wedge x_{22})$$

- Defina valores para cada componente do vetor  $x$ , para que a função  $F(x)$  seja avaliada como VERDADEIRA.

## O tamanho do espaço de buscas pode ser grande demais

- Quantas diferentes atribuições existem para as componentes do vetor  $x$ ?
- Em outras palavras, qual é o tamanho do espaço de busca do problema SAT?

$$F(x) = \overline{(x_{89} \vee x_{78} \wedge x_{01})} \wedge \overline{(x_{10} \vee x_{99})} \vee \dots \wedge (x_{56} \wedge x_{22})$$

## O tamanho do espaço de buscas pode ser grande demais

$$F(x) = \overline{(x_{89} \vee x_{78} \wedge x_{01})} \wedge \overline{(x_{10})} \vee x_{99}) \vee \dots \wedge (x_{56} \wedge x_{22})$$

- Qual é o tamanho do espaço de busca ?

$$|S| = 2^{|x|}$$

$$|S| = 2^{100} \approx 10^{30}$$

10.000.000.000.000.000.000.000.000.000

atribuições diferentes

## O tamanho do espaço de buscas pode ser grande demais

- Qual é o tamanho do espaço de busca ?

$$|S| = 2^{100} \approx 10^{30}$$

10.000.000.000.000.000.000.000.000.000

atribuições diferentes

- Suponha que temos um computador que consegue avaliar **1000** atribuições diferentes por segundo.

## O tamanho do espaço de buscas pode ser grande demais

- Qual é o tamanho do espaço de busca ?

$$|S| = 2^{100} \approx 10^{30}$$

10.000.000.000.000.000.000.000.000.000

atribuições diferentes

- Suponha que temos um computador que consegue avaliar 1000 atribuições diferentes por segundo.
- Se este computador estivesse funcionando desde o BIG BANG (a 15 bilhões de anos), ele não teria analisado nem 1% de todas as possibilidades.

# SAT

- O SAT foi o primeiro problema a ser classificado como NP-completo por Cook no ano de 1971;

# SAT

- O SAT foi o primeiro problema a ser classificado como NP-completo por Cook no ano de 1971;
- A SAT é um problema especial, pois todos os problemas que possuem algoritmos (polinomiais ou não) podem ser transformados no problema da SAT.

# SAT

- O SAT foi o primeiro problema a ser classificado como NP-completo por Cook no ano de 1971;
- A SAT é um problema especial, pois todos os problemas que possuem algoritmos (polinomiais ou não) podem ser transformados no problema da SAT.
  - Prova usa definição matemática da **Máquina de Turing não-determinística** (MTND), capaz de resolver qualquer problema em NP. Incluindo uma descrição da máquina e de como instruções são executadas em termos de fórmulas booleanas.

*Estabelece uma correspondência entre todo problema em NP (expresso por um programa na MTND) e alguma instância de SAT.*

# SAT

- O SAT foi o primeiro problema a ser classificado como NP-completo por Cook no ano de 1971;
- A SAT é um problema especial, pois todos os problemas que possuem algoritmos (polinomiais ou não) podem ser transformados no problema da SAT.
  - Prova usa definição matemática da Máquina de Turing não-determinística (MTND), capaz de resolver qualquer problema em NP. Incluindo uma descrição da máquina e de como instruções são executadas em termos de fórmulas booleanas.

*Estabelece uma correspondência entre todo problema em NP (expresso por um programa na MTND) e alguma instância de SAT.*
- Para completar o raciocínio, veremos a redução de problemas.

# Redução de Problemas ou Transformação Polinomial

## Introdução

# NP-Completo

- Informalmente, um problema está em NP-Completo se ele está em NP e é “tão difícil” quanto qualquer outro problema em NP
- Esta noção de ser “tão difícil quanto” está ligada ao conceito de transformação/redução polinomial

# Redução de Problemas

- Ex.:
  - *Podemos resolver uma equação de primeiro grau através de um algoritmo que resolve equações de segundo grau...*

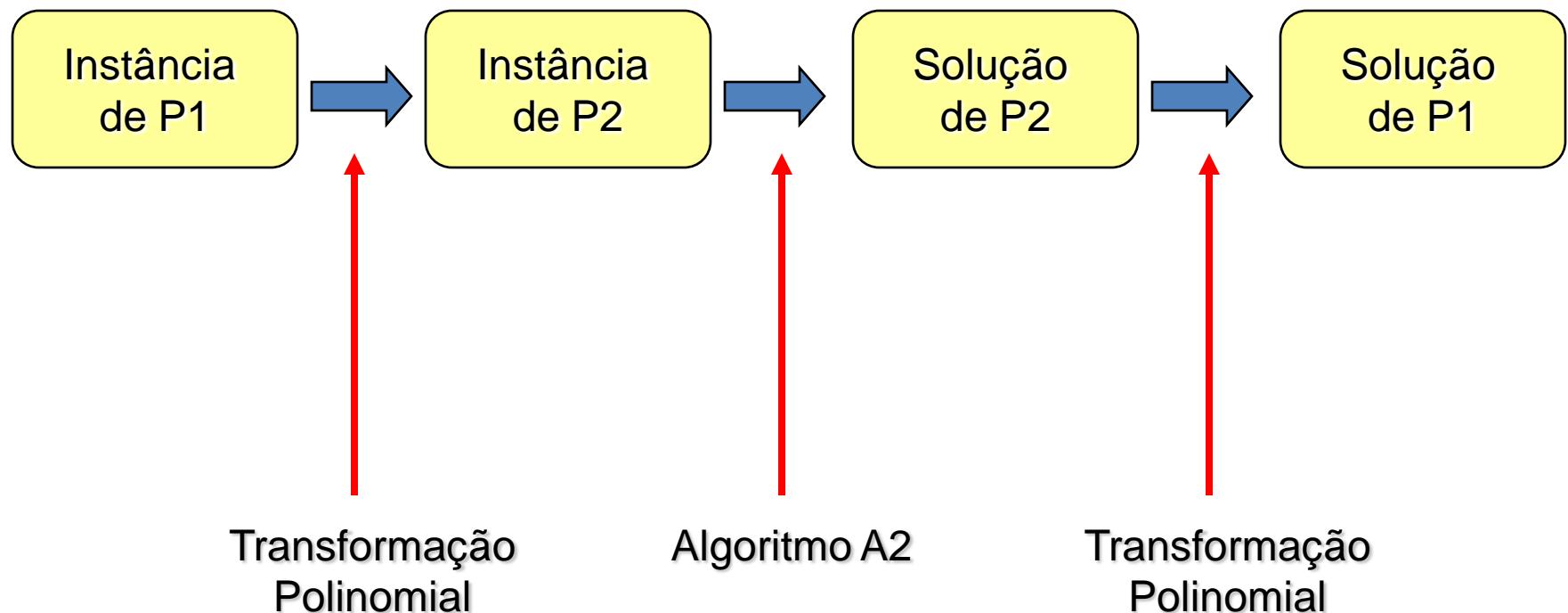
# Redução de Problemas

- Ex.:
  - Podemos resolver uma equação de primeiro grau através de um algoritmo que resolve equações de segundo grau...

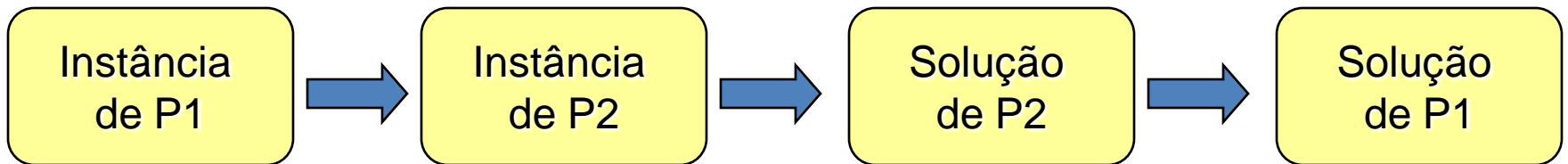
$$\begin{array}{c} 45x + 10 = 0 \\ \downarrow \\ 0x^2 + 45x + 10 = 0 \end{array}$$

- Ou seja, as equações de primeiro grau são um caso particular das equações de segundo grau.

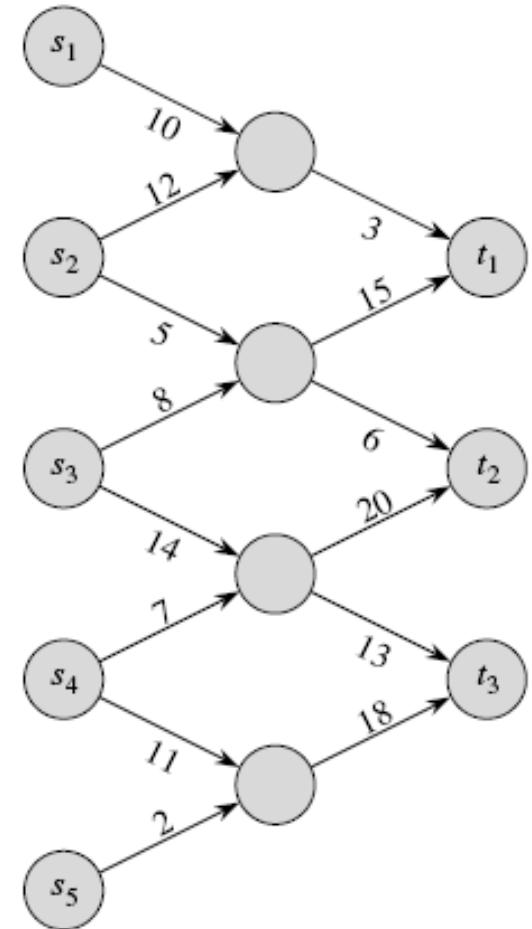
# Transformação Polinomial



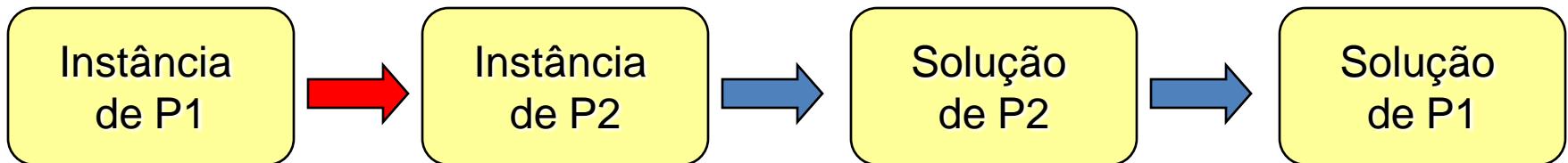
# Transformação Polinomial



- Exemplo:
  - P1: Desejamos calcular o **fluxo máximo entre um conjunto de nós servidores e um conjunto de nós consumidores**;
  - Não conhecemos algoritmo direto e simples para este problema...

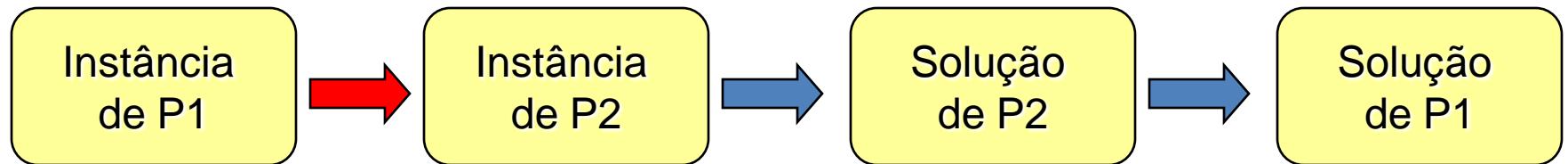


# Transformação Polinomial

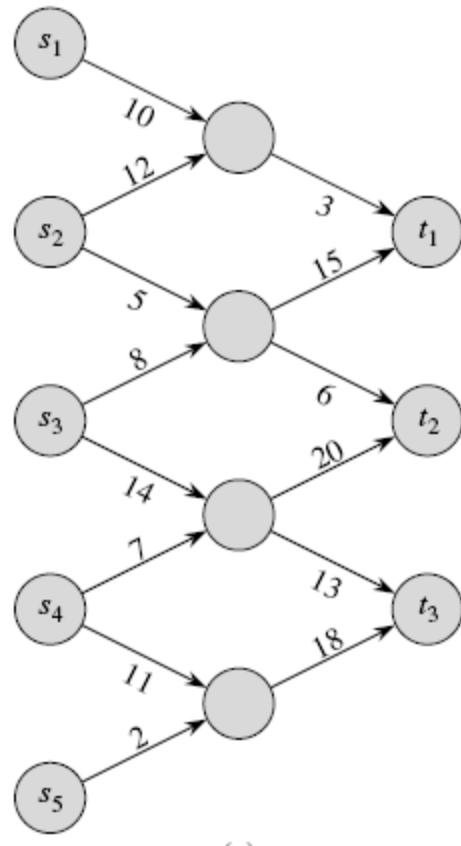


- Exemplo:
  - Podemos transformar a instância  $G=(V,A)$  em uma instância modificada  $G'=(V',A')$  para problema de fluxo máximo de única origem e único destino, onde conhecemos algoritmo;
  - $G'=(V',A')$ , onde:
    - $V'=V \cup \{s,t\}$
    - $A''=\{(s,x, \infty) \mid x \text{ é origem em } G\}$
    - $A'''=\{(y,t, \infty) \mid y \text{ é destino em } G\}$
    - $A'=A \cup A'' \cup A'''$

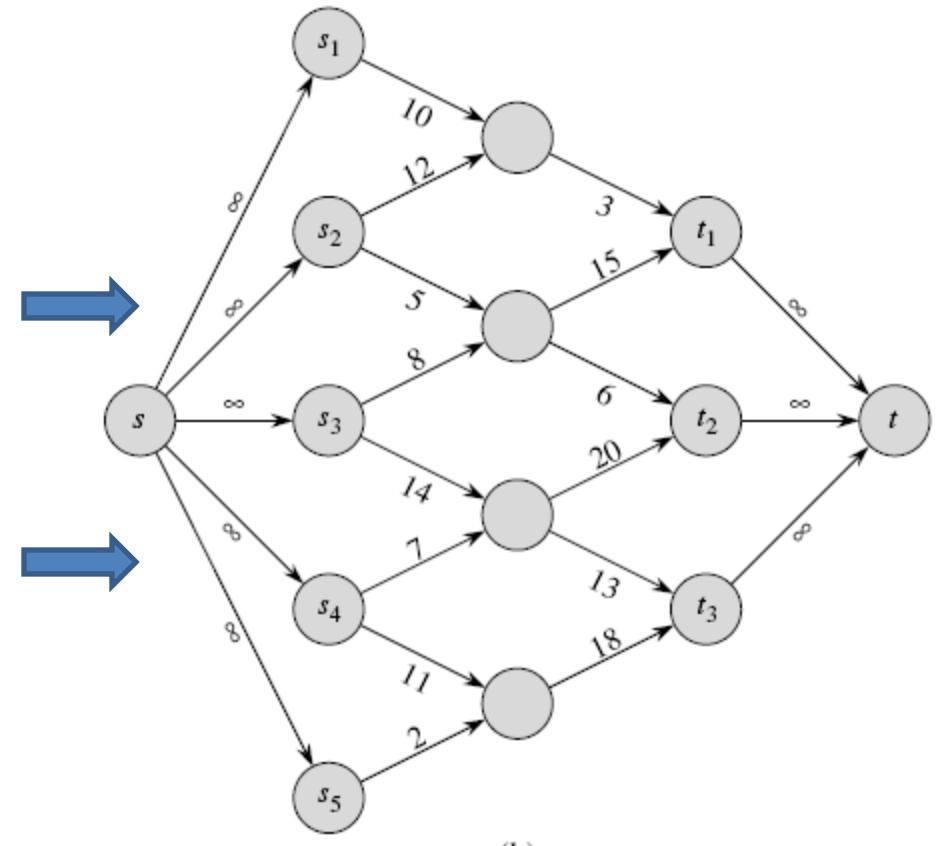
# Transformação Polinomial



- Exemplo:

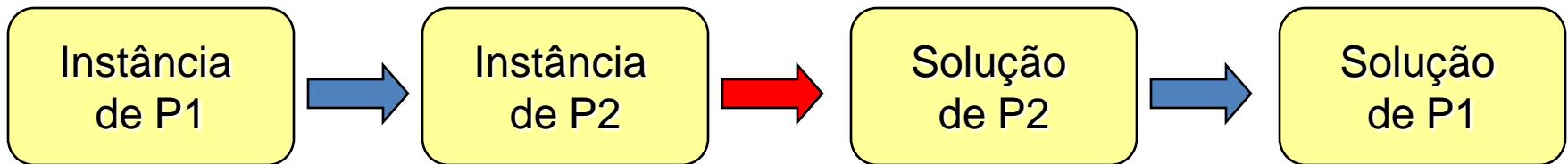


(a)

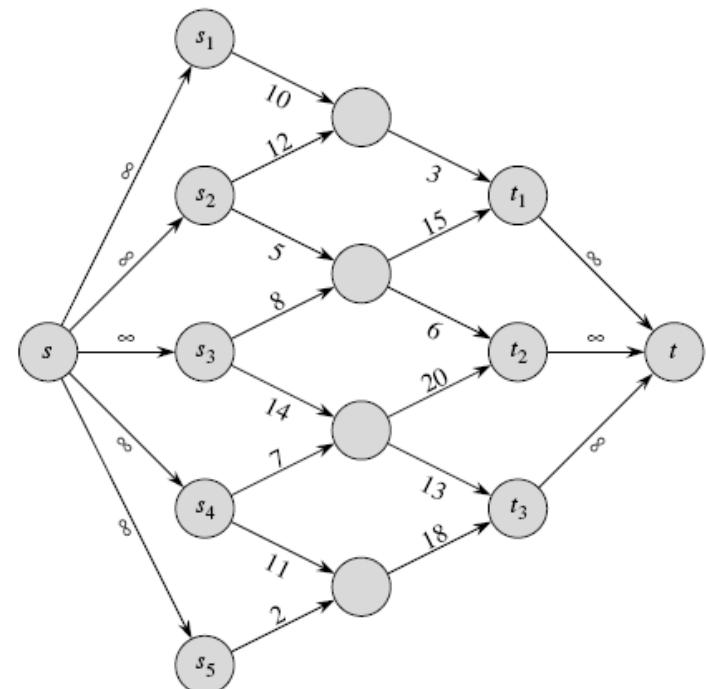


(b)

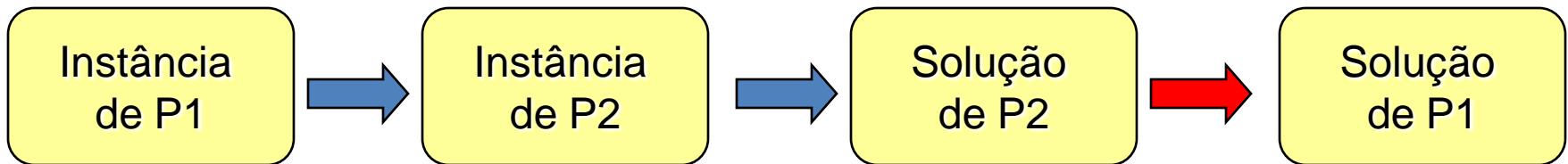
# Transformação Polinomial



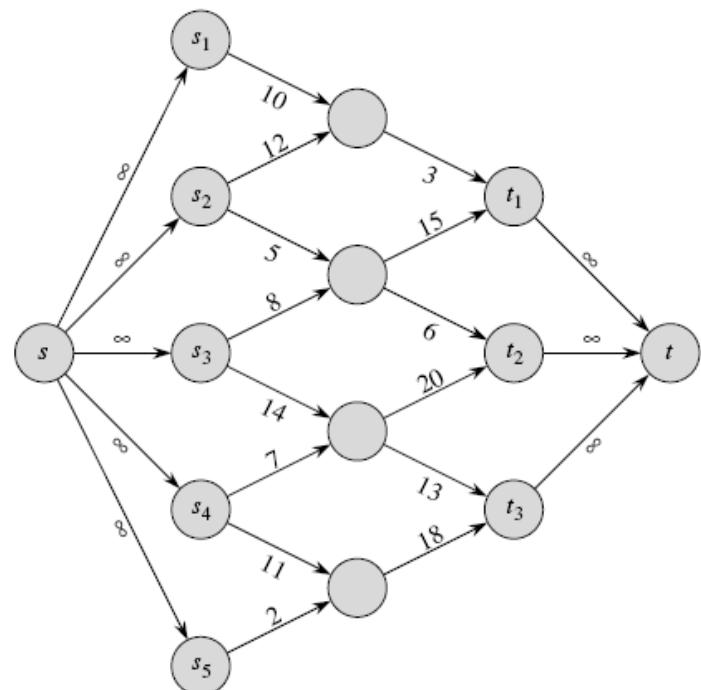
- Exemplo:
  - Com a instância de P2, podemos agora aplicar o algoritmo de fluxo máximo conhecido em P2: Por exemplo, Ford-Fulkerson;



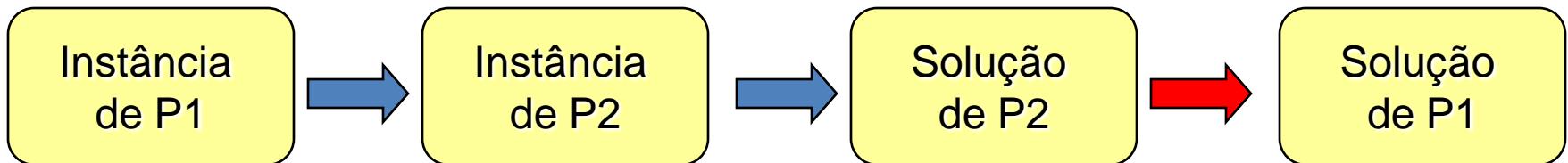
# Transformação Polinomial



- Exemplo:
  - Obtendo a solução de P2, podemos convertê-la em solução de P1, eliminando as arestas e nós criados artificialmente para P2.
  - Assim, conhecemos o fluxo máximo de P1.



# Transformação Polinomial



- Considerações:
  - Este exemplo mostrado, é a **transformação de um problema polinomial em outro polinomial**;
  - Ou seja, a **transformação polinomial não serve apenas para o estudo da classe de problemas NP-Completo**;
  - Transformando o fluxo máximo de várias origens e destinos (P1) no fluxo máximo de origem e destino único (P2), estamos dizendo:
    - P2 é no mínimo tão difícil quanto P1.

# Redução de Problemas

- *Se qualquer problema NP-Completo for reduzido em tempo polinomial a um problema qualquer P, então temos indício que P é NP-Completo*

# Redução de Problemas

- *Se qualquer problema NP-Completo for reduzido em tempo polinomial a um problema qualquer P, então, P é NP-Completo*
- Vamos juntar os fatos:
  - Todos os problemas que conhecemos algoritmos podem ser transformados na SAT;

# Redução de Problemas

- *Se qualquer problema NP-Completo for reduzido em tempo polinomial a um problema qualquer P, então, P é NP-Completo*
- Vamos juntar os fatos:
  - Todos os problemas que conhecemos algoritmos podem ser transformados na SAT;
  - Se transformamos, por exemplo, a SAT em um problema P, então:
    - P é pelo menos tão difícil quanto a SAT;

# Redução de Problemas

- *Se qualquer problema NP-Completo for reduzido em tempo polinomial a um problema qualquer P, então, P é NP-Completo*
- Vamos juntar os fatos:
  - Todos os problemas que conhecemos algoritmos podem ser transformados na SAT;
  - Se transformamos, por exemplo, a SAT em um problema P, então:
    - P é pelo menos tão difícil quanto a SAT;
    - E a SAT é pelo menos tão difícil quanto P;

# Redução de Problemas

- *Se qualquer problema NP-Completo for reduzido em tempo polinomial a um problema qualquer P, então, P é NP-Completo*
- Vamos juntar os fatos:
  - Todos os problemas que conhecemos algoritmos podem ser transformados na SAT;
  - Se transformamos, por exemplo, a SAT em um problema P, então:
    - P é pelo menos tão difícil quanto a SAT;
    - E a SAT é pelo menos tão difícil quanto P;
    - Então, ambos possuem a mesma complexidade computacional;

# Redução de Problemas

- O PCV já foi mostrado ser NP-Completo.

# Redução de Problemas

- O PCV já foi mostrado ser NP-Completo.
- Se temos o Problema P1, e **reduzimos o PCV a P1**, isso quer dizer que:

# Redução de Problemas

- O PCV já foi mostrado ser NP-Completo.
- Se temos o Problema P1, e **reduzimos o PCV a P1, isso quer dizer que:**
  - P1 é NP-Completo;

# Redução de Problemas

- O PCV já foi mostrado ser NP-Completo.
- Se temos o Problema P1, e **reduzimos o PCV a P1**, isso quer dizer que:
  - P1 é NP-Completo;
  - E mais importante do que isso:
    - *Se algum dia alguém encontrar algum algoritmo polinomial para qualquer problema NPC, todos os problemas da classe NPC são também resolvidos em tempo polinomial;*

# Redução de Problemas

- O PCV já foi mostrado ser NP-Completo.
- Se temos o Problema P1, e **reduzimos o PCV a P1**, isso quer dizer que:
  - P1 é NP-Completo;
  - E mais importante do que isso:
    - *Se algum dia alguém encontrar algum algoritmo polinomial para qualquer problema NPC, todos os problemas da classe NPC são também resolvidos em tempo polinomial;*
  - É a famosa tentativa de mostrar que  $P=NP$ ;

# Conclusões Parciais

- Quase ninguém acredita que P=NP;
- **Atualmente, são conhecidos mais de 3.000 problemas NP-Completo**, e para nenhum deles, foi encontrado algoritmo polinomial;
  - Este é um forte indício de que as classes são realmente distintas;
- **Acredita-se também que NPC seja muito maior do que P.**

# NP-Completo

*Complementando...*

# NP-Completo

- **Para provar que um problema é NP-Completo**, são necessários **dois passos**:
  - 1) Mostre que o **problema está em NP**:
    - apresentando pelo menos um algoritmo não determinístico polinomial para o problema; ou
    - Verificando sua solução em tempo polinomial;
  - 2) Mostre que **pelo menos um problema NP-Completo pode ser reduzido diretamente ao problema estudado**.
    - SAT ou qualquer outro NP-Completo.

# NP-Completo

- Um problema  $P_1$  é NP-Completo se:
  - 1  $P_1 \in \text{NP}$
  - 2  $P' \leq P_1$  para todo  $P' \in \text{NP}$

Podemos simplificar propriedade 2 para:

**2' existe  $P' \in \text{NP-Completo}$  tal que  $P' \leq P_1$**

(Veja prova do Lema 34.8 no livro do Cormem)

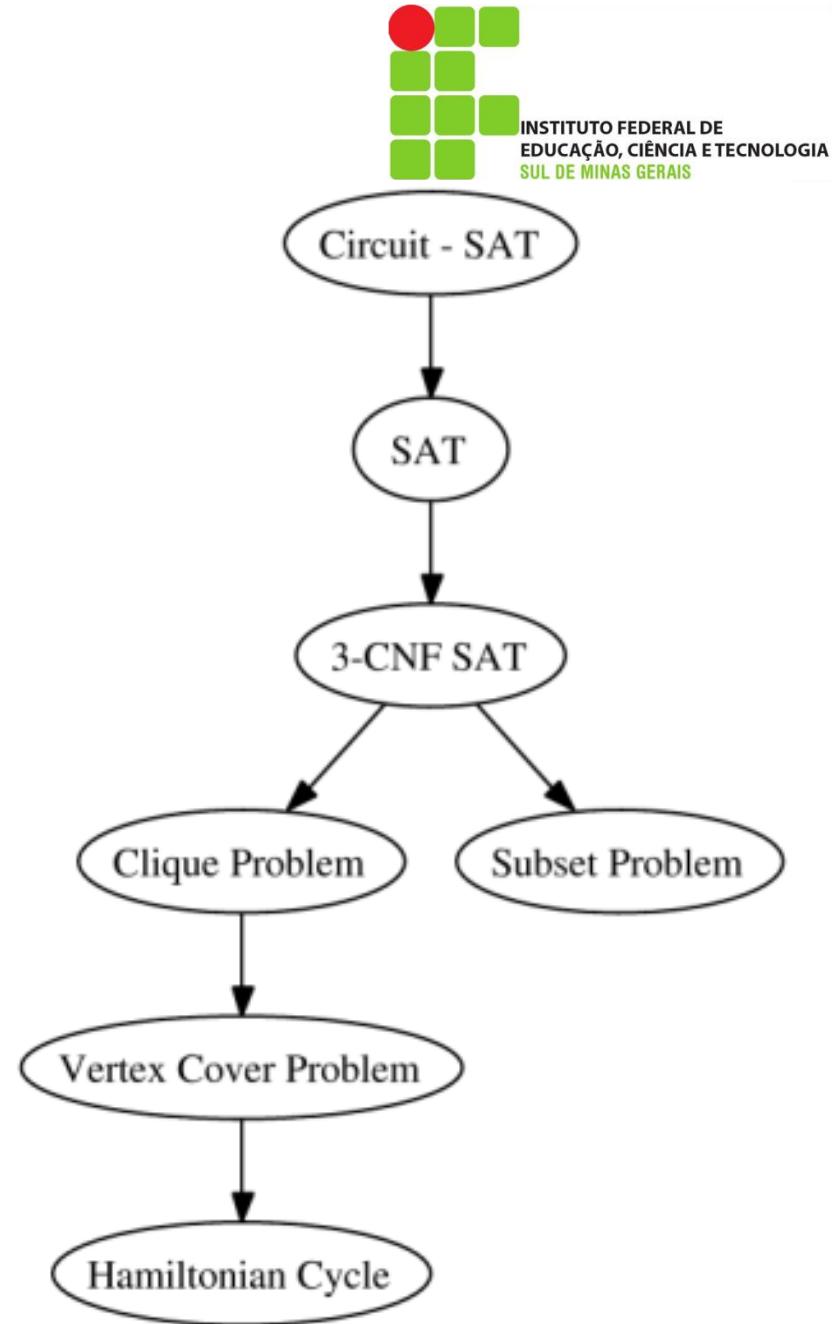
- Um problema  $P_1$  é NP-Difícil se somente a propriedade 2 (ou 2') for válida

# NP-Completo

- Isso só é possível porque Cook em 1971 apresentou uma prova direta da SAT ser um problema NP-Completo, além do fato de que a redução polinomial é transitiva:
  - Se  $(SAT \propto P1)$  e  $(P1 \propto P2)$  então
    - $SAT \propto P2$
  - Como temos a prova de que qualquer instância de qualquer problema com algoritmo pode ser transformado em uma instância da SAT, então,  $P2 \propto SAT$ .

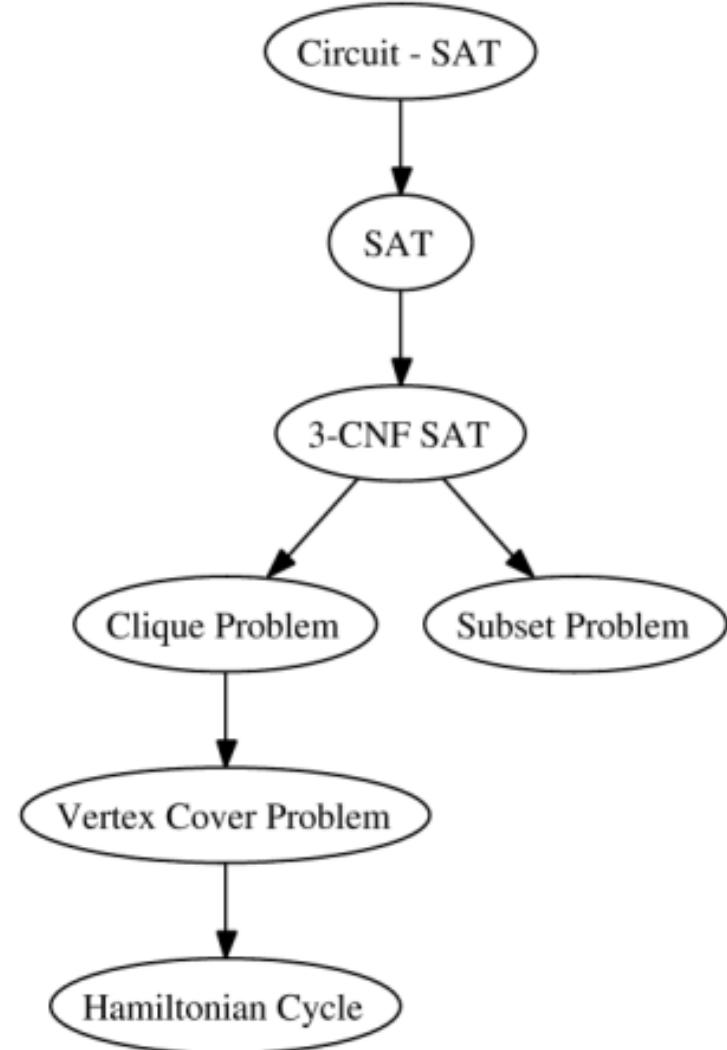
# NP-Completo

- Vamos supor que conhecemos os seguintes problemas na classe NP-Completo:
- A seta do problema P para o problema Q, indica que P pode ser reduzido em tempo polinomial ao problema Q.



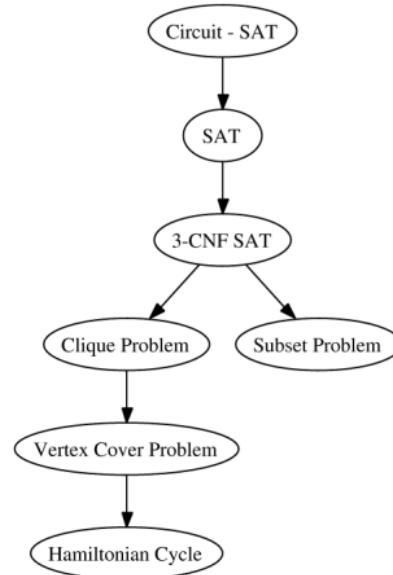
# NP-Completo

- Como mostrar que o Problema do Caixeiro Viajante (PCV) é NP-Completo?
- Quais são mesmo os passos para a prova?



# NP-Completo

- Como mostrar que o PCV é NP-Completo?
- Quais são mesmo os passos para a prova?
  - 1: Mostrar que existe algoritmo ND para o PCV;
  - 2: Reduzir qualquer problema NP-Completo para o PCV;



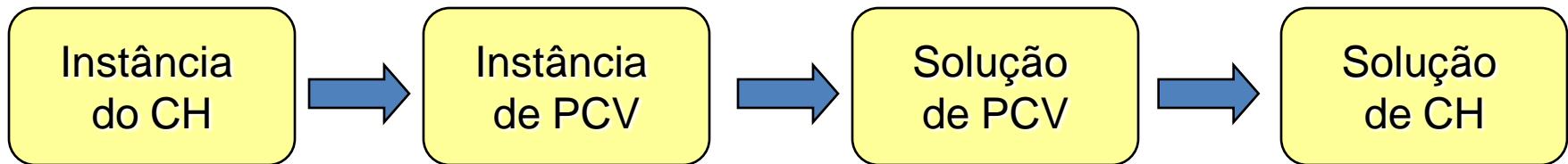
# NP-Completo

- 1: Mostrar que existe algoritmo ND para o PCV;
- Algoritmo ND para o PCV:

```
resolvePCV(V,A,k)
{
    int i = 0;
    S = {V[i]};
    for (i = 0; i < |V|; i++)
    {
        j = escolhe (V[i], lista-adj(V[i]));
        S = S ∪ V[j]
    }
    return verificaPCV (V,A,k,S);
}
```

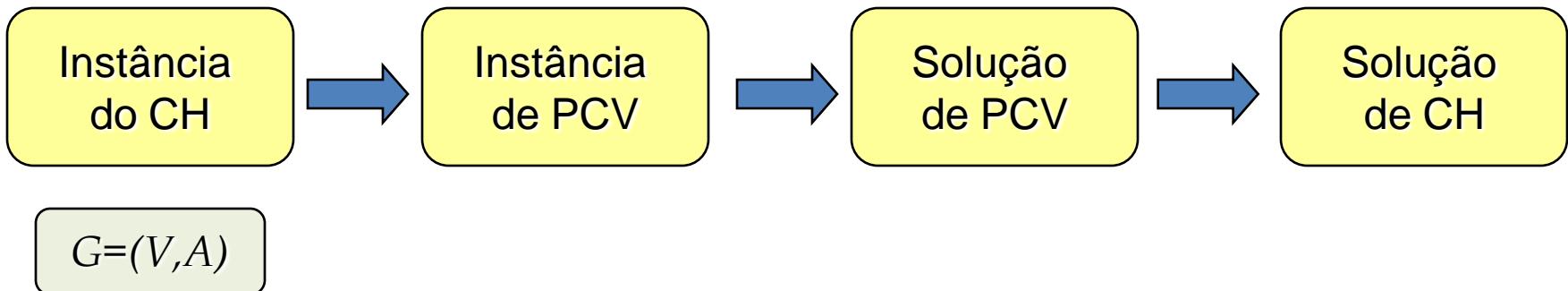
# NP-Completo

- 2: Reduzir qualquer problema NP-Completo para o PCV;
  - Vamos **reduzir o Ciclo Hamiltoniano no PCV**;

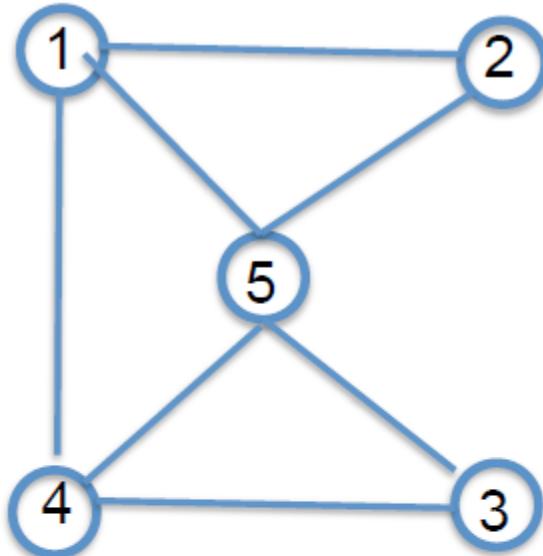


# NP-Completo

- 2: Reduzir qualquer problema NP-Completo para o PCV;
  - Vamos **reduzir o Ciclo Hamiltoniano no PCV**;



# NP-Completo



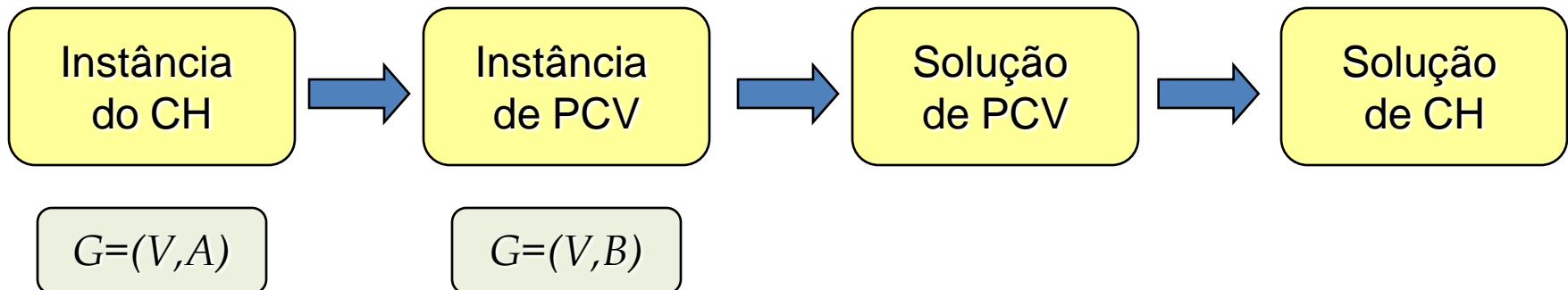
?

Exemplo de Instância de CH  
 $G(V, A)$

Instância de PCV  
 $G'(V', A')$   
 $k$

# NP-Completo

- 2: Reduzir qualquer problema NP-Completo para o PCV;
  - Vamos reduzir o Ciclo Hamiltoniano no PCV;

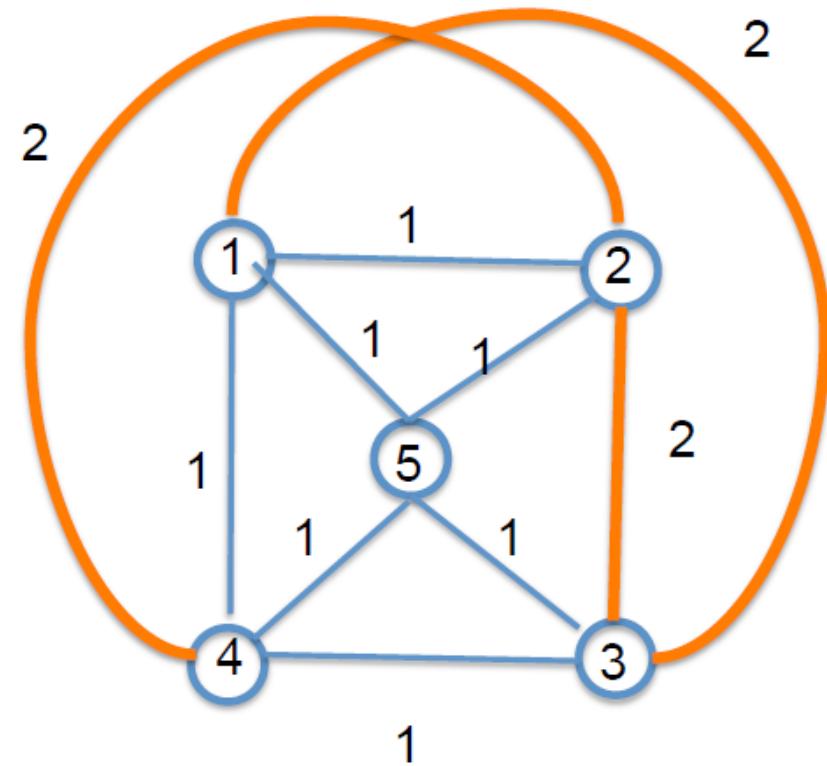
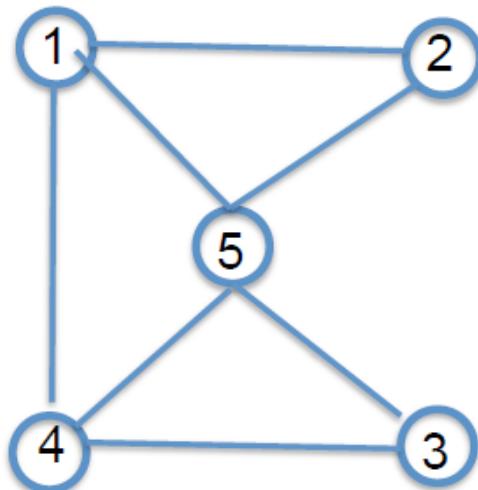


$$B = \{(i, j) \mid i, j \in V\}$$

$$c : V \times V \rightarrow \mathbb{R}$$

$$c(i, j) = \begin{cases} 1, & \text{se } (i, j) \in A \\ 2, & \text{se } (i, j) \notin A \end{cases}$$

# NP-Completo

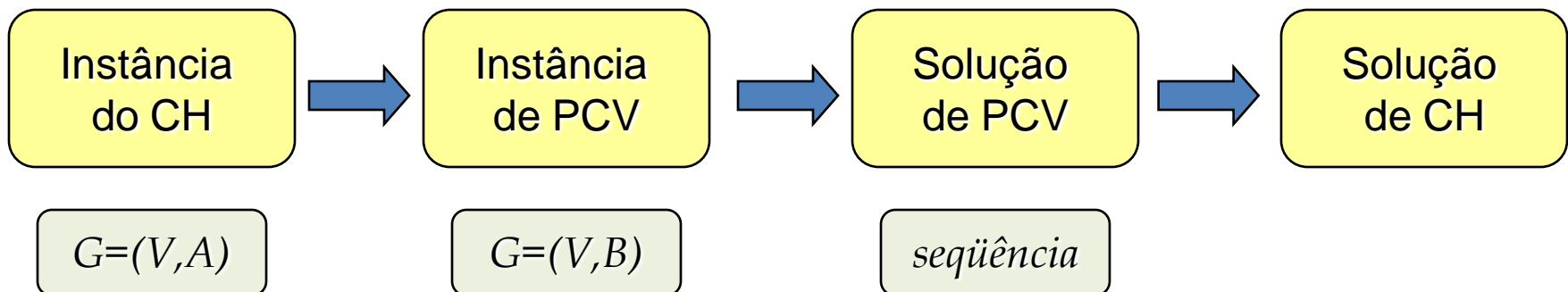


Exemplo de Instância de CH  
 $G(V, A)$

Instância de PCV  
 $G'(V', A')$   
 $k = |V| = 5$

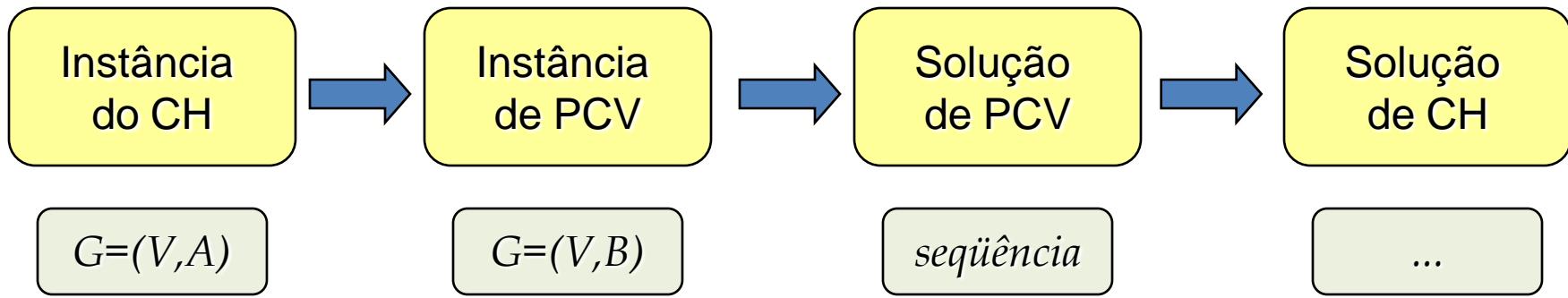
# NP-Completo

- 2: Reduzir qualquer problema NP-Completo para o PCV;
  - Vamos reduzir o Ciclo Hamiltoniano no PCV;



# NP-Completo

- 2: Reduzir qualquer problema NP-Completo para o PCV;
  - Vamos reduzir o Ciclo Hamiltoniano no PCV;

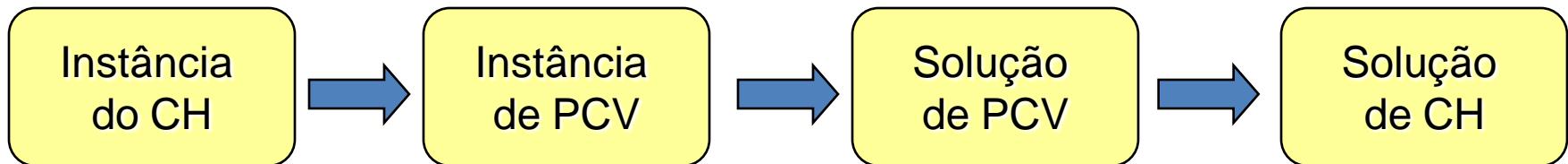


Se o custo do caminho do PCV é  $|V|$  então,  
existe CH para  $G$ .

Se o custo do caminho do PCV é maior que  $|V|$   
então não existe CH para  $G$ .

# NP-Completo

- 2: Reduzir qualquer problema NP-Completo para o PCV;
  - Vamos **reduzir o Ciclo Hamiltoniano no PCV**;



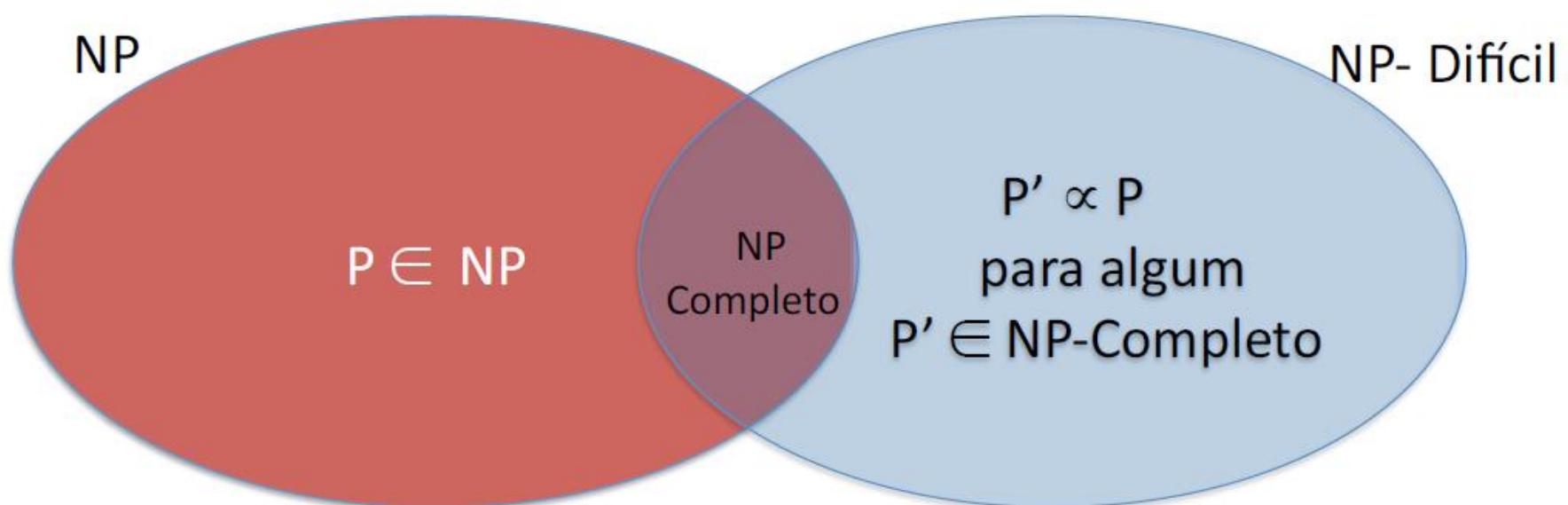
- Ao resolver um problema NP-Completo através de outro problema Y, então Y também é NP-Completo, pois ele é no mínimo tão difícil quanto o problema NP-Completo.

# A classe NP-Difícil

# NP-Difícil

- Também conhecida como **NP-Árduo (NP-Hard)**;
- Para mostrar que um problema é **NP-Difícil, basta satisfazer a seguinte condição:**
  - Mostre que pelo menos um problema NP-Completo pode ser reduzido diretamente ao problema estudado.
- Lembrando que esta é uma condição necessária para o problema ser NP-Completo, ou seja:
  - **TODO PROBLEMA NP-COMPLETO É TAMBÉM NP-DIFÍCIL.**

# NP-Difícil



# NP-Difícil

## Exemplo de Problema exclusivamente NP-Difícil

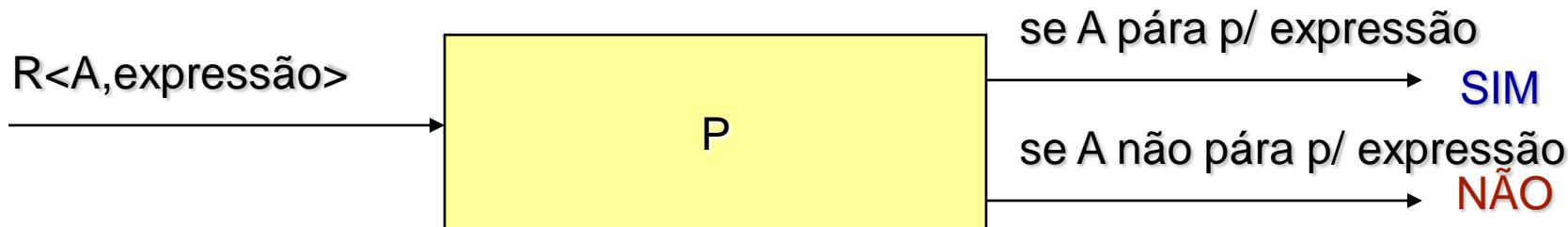
- **Problema da Parada**
  - Para mostrar que um problema é exclusivamente NP-Difícil, quais são os passos?
    - 1) Mostre que o problema *não* está em NP:
      - Mostrando que não existe nenhum algoritmo para ele;
    - 2) Mostre que pelo menos um problema NP-Completo pode ser reduzido diretamente ao problema estudado.

# NP-Difícil

## Exemplo de Problema exclusivamente NP-Difícil

- Problema da Parada: mostrando SAT pode ser reduzida ao PP

- Considere o algoritmo  $A$  cuja entrada é uma expressão booleana na forma normal conjuntiva com  $n$  variáveis;
- Basta tentar  $2^n$  possibilidades e verificar se é satisfável.
  - Se a expressão for SAT,  $A$  pára;
  - Senão, entra em *loop* (não pára).



- Logo, o problema da parada é NP-difícil, mas não é NP-completo.

# NP-Difícil

## Exemplo de Problema exclusivamente NP-Difícil

- Problema da Parada:
  - O problema da parada é no mínimo tão difícil quanto o problema da SAT.
  - O problema da parada é no mínimo tão difícil quanto qualquer problema.

# NP-Difícil

- Podemos mostrar que o problema da parada pode ser reduzido a outros problemas;
  - O que isso quer dizer?
  - Em que classe de problemas este outro problema está?

# Bibliografia

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; (2002). Algoritmos – Teoria e Prática. Tradução da 2<sup>a</sup> edição americana. Rio de Janeiro. Editora Campus.
- TAMASSIA, ROBERTO; GOODRICH, MICHAEL T. (2004). Projeto de Algoritmos - Fundamentos, Análise e Exemplos da Internet.
- ZIVIANI, N. (2007). Projeto e Algoritmos com implementações em Java e C++. São Paulo. Editora Thomson;
- Material de aulas do Professor Loureiro (DCC-UFMG)
  - <http://www.dcc.ufmg.br/~loureiro/paa/>

